**SOCIAPP**

Hamza belyahioui, Fadel Biaou, Abdoul-waris Konate

**Documentation**

Sommaire

**Introduction**

* Résumé du projet
* Diagramme de basse de donnes
* Outils
* Artchitecture cloud
* Development de l’Api
* Perfermanbce et Scalability
* Mise en place de azure
* Point de amélioration
* Conculusion
* Résumé du projet

Développez une plateforme sociale "headless" axée sur la création et la gestion de contenu via des API. Cette plateforme servira de **backend pour des réseaux sociaux** et permettra une intégration transparente avec différentes applications frontales.$

* Fonctionnalité principale

1. **Authentification et autorisation des utilisateurs** :
   * Gestion des comptes utilisateurs (créateurs).
   * Options de confidentialité pour rendre un profil public ou privé.
2. **Création et gestion de contenu** :
   * Publication, modification, suppression et récupération de contenu.
   * Stockage sécurisé du contenu utilisateur.
3. **Gestion des médias** :
   * Téléchargement et récupération d’images et de vidéos.
4. **Commentaires et modération** :

* Permettre les commentaires sur les publications et leur modération.

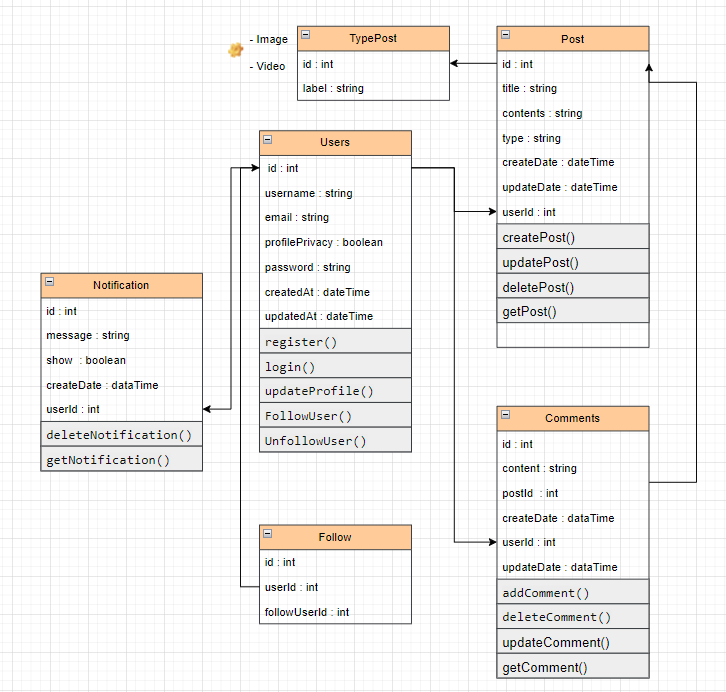
1. **Fonctionnalité de recherche** :

* Rechercher des publications par utilisateur ou par mot-clé.

1. **Notifications** :

Notifier les abonnés des nouvelles publications.

* Diagramme de base de données



**Users** :

Représente les utilisateurs avec leurs informations personnelles (nom, email, etc.) et des actions comme s'inscrire, se connecter, ou suivre d'autres utilisateurs.

**Post** :

Contient les publications créées par les utilisateurs, avec des actions pour créer, mettre à j our ou supprimer des posts.

**Comments** :

Permet aux utilisateurs de commenter les posts, avec des fonctionnalités pour ajouter, supprimer ou modifier un commentaire.

**TypePost** :

Définit le type de publication (image, vidéo, etc.), associé à chaque post.

**Notification** :

Gère les notifications des utilisateurs, comme les messages ou alertes.

**Follow** :

Gère les relations entre les utilisateurs (qui suit qui)

* Outils

1. **Une image contenant Graphique, violet, Caractère coloré, symbole

   Description générée automatiquementVisual Studio**

est un environnement de développement intégré (IDE) de Microsoft, et ASP.NET est un framework de développement web utilisé pour construire des applications web robustes et dynamiques. Ensemble, Visual Studio et ASP.NET offrent un puissant ensemble d'outils pour le développement d'applications web. Voici quelques aspects clés

1. **Github**

Une image contenant logo, Police, Graphique, symbole

Description générée automatiquementPour la gestione de projet on a utilise Github

GitHub est une plateforme web qui utilise Git pour le contrôle de version et la gestion de projets

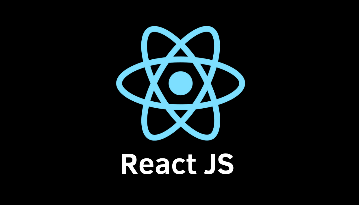
1. **Azure**

**Microsoft Azure** est une plateforme de services cloud complète qui permet de créer, déployer et gérer des applications

1. Une image contenant Bleu électrique, Graphique, bleu, capture d’écran

   Description générée automatiquement**.Net**

Utiliser **.NET avec Azure** est idéal car Azure est parfaitement optimisé pour les applications .NET, offrant une intégration native avec des services comme **Azure App Service, Azure Functions**, et **Azure SQL**, ce qui simplifie le déploiement et la gestion des applications. De plus.

5**.React**

Utiliser **.** **NET avec Azure** est idéal car Azure est parfaitement optimisé pour les applications .NET, offrant une intégration native avec des services comme **Azure App Service, Azure Functions**, et **Azure SQL**, ce qui simplifie le déploiement et la gestion des applications. De plus.

* Une image contenant capture d’écran, diagramme, texte, ligne

  Description générée automatiquementArchitecture Cloud

**User (Utilisateur)**

C'est l'utilisateur final qui interagit avec l'application web via un navigateur ou un appareil.

**Font**

Représente Api web, hébergée sur Azure App Service.

**Api**

Représente Api web, hébergée sur Azure App Service.

**GitHub**

Source du code de l'application.

Avec Azure, vous pouvez configurer une intégration CI/CD (Continuous Integration/Continuous Deployment) via GitHub

**Storage**

Utilisé pour stocker des fichiers, images des videos

**MySQL DB**

Base de données relationnelle utilisée par l'application pour stocker les données, avec Azure Database for MySQL

**Monitor - logs**

………………

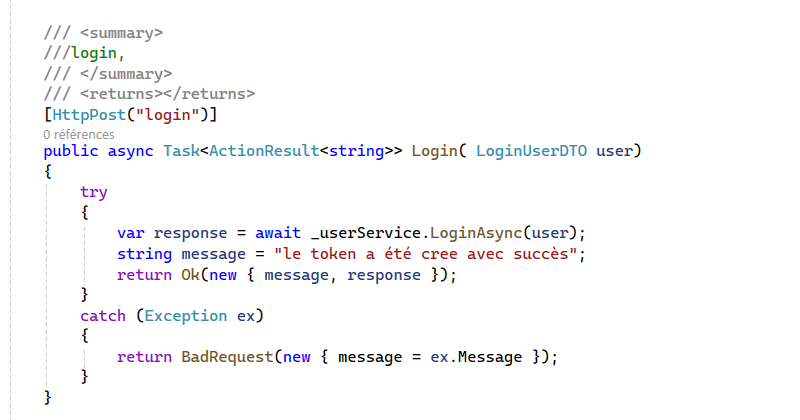
* Development de l’API

J'ai organisé mon API en deux couches principales :

1. **Controller** : C'est la porte d'entrée où les requêtes arrivent depuis le frontend.
2. **Service** : Ici, j'ai mis toute la logique métier pour éviter d'avoir du code complexe dans le controller.

Quand quelqu’un veut se connecter :

**Controller (Login)** :

* + Le frontend envoie l'email et le mot de passe à l'API.
  + Le controller appelle la méthode LoginAsync de la couche service et récupère la réponse.
  + Si tout va bien, je retourne un token (JWT) au frontend.
  + Si une erreur se produit, j'envoie un message d'erreur.

**Service (LoginAsync)** :

* + Je commence par chercher l’utilisateur dans ma base de données via l'email.
  + Si l’utilisateur n’existe pas ou si le mot de passe est incorrect, je retourne une erreur.
  + Sinon, je crée un **token JWT** contenant des informations comme l'ID utilisateur et son email.
  + Une image contenant texte, capture d’écran, Police

    Description générée automatiquementEnfin, je retourne ce token.

1. **Créer un Token** :
   * Le token est créé avec une clé secrète définie dans la configuration.
   * Il contient des informations comme l’émetteur, le destinataire, et une date d’expiration



le token crée je vais le utilise pour sécurise les Endpoint :



j’ai utilisé Authorize pour protéger les routes comme celle qui récupère les posts des utilisateurs. Ça permet de s’assurer que seules les personnes authentifiées, c'est-à-dire celles qui envoient un jeton JWT valide, peuvent y accéder. Quand une requête arrive, ASP.NET vérifie automatiquement le jeton : est-il valide, signé correctement, et non expiré ? Si tout est bon, l’accès est autorisé, sinon la requête est rejetée avec un message d’erreur. En gros, [Authorize] garantit que les données sensibles restent sécurisées et accessibles uniquement aux utilisateurs connectés.

Pour enregistrer un utilisateur :

1. Le frontend envoie les informations de l'utilisateur (nom, email, mot de passe, etc.).
2. Dans le controller, je valide ces données en appelant la méthode RegisterAsync de la couche service.
3. Une image contenant texte, capture d’écran, Police

   Description générée automatiquementSi tout va bien, l'utilisateur est ajouté à la base de données.

**Étape 4 : Déployer sur Azure avec GitHub Actions**

1. **Déploiement Azure** :
   * J’ai créé un **App Service** dans Azure pour héberger mon API.
   * J’ai ajouté la clé secrète JWT, l’émetteur et le destinataire dans les **paramètres de l’App Service**.
2. **GitHub Actions** :
   * J'ai configuré GitHub Actions pour automatiser le déploiement.
   * Chaque fois que je pousse une modification sur la branche principale, GitHub déploie automatiquement l'API sur Azure.

Avec cette API, j’ai réussi à :

1. Créer un système d’authentification basé sur JWT pour sécuriser mon API.
2. Permettre l’enregistrement et la connexion d’utilisateurs.
3. Automatiser le déploiement dans Azure via GitHub Actions.

Cela m'assure que l'API est toujours à jour sans intervention manuelle après chaque mise à jour de code.

* Mise en place du Azure

**Api :**

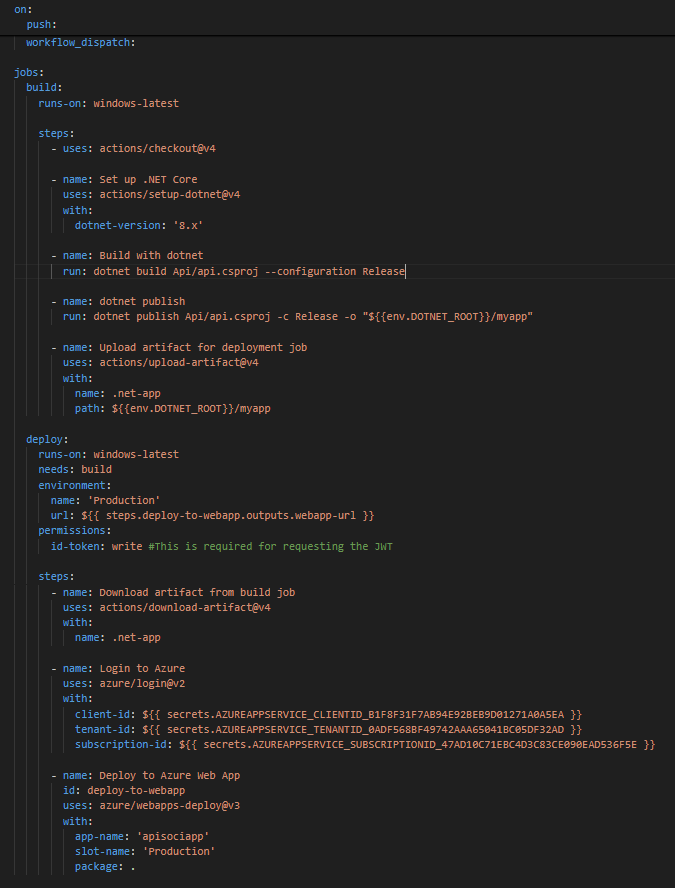
**Création de mon App Service**

Je commence par me connecter au portail Azure, puis je me rends dans la section "App Services" pour créer une nouvelle application.

**Configuration des paramètres nécessaires**

Une fois l'App Service créé, je configure la connexion avec mon dépôt GitHub. Cela permet de générer automatiquement un pipeline dans mon workflow. Ce pipeline sera déclenché chaque fois que je réaliserai un commit sur la branche main.

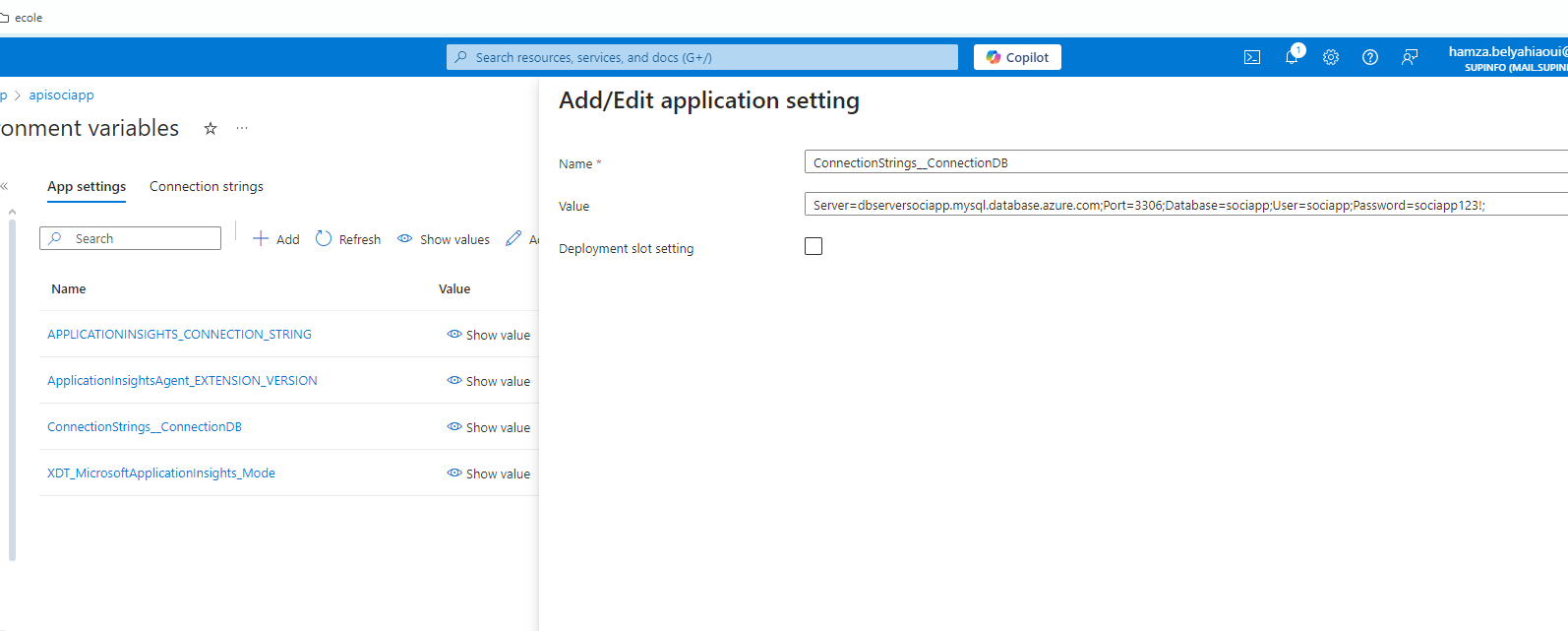
Ainsi, après chaque commit, le pipeline s'occupera de la construction (build) de mon API, puis procédera au déploiement sur l'App Service. Une fois le déploiement

terminé, je peux tester mon API en utilisant l'URL fournie par l'App Service.

**DB :**

J'ai créé une base de données dans Azure en utilisant Azure Database for MySQL. Une fois la base de données mise en place, je l'ai liée à mon application web hébergée dans un App Service sur Azure. Pour ce faire, j'ai récupéré la chaîne de connexion de ma base de données MySQL dans le portail Azure. Cette chaîne contient toutes les informations nécessaires, telles que l'hôte, le nom d'utilisateur, le mot de passe et le nom de la base de données.

Ensuite, j'ai ajouté cette chaîne de connexion dans les paramètres de l'App Service de mon application. J'ai utilisé les variables d'environnement pour stocker cette chaîne de connexion afin qu'elle soit disponible au moment de l'exécution de mon API .NET. Cela permet à l'API d'interagir avec la base de données de manière sécurisée et flexible, sans exposer directement les informations sensibles dans le code source.

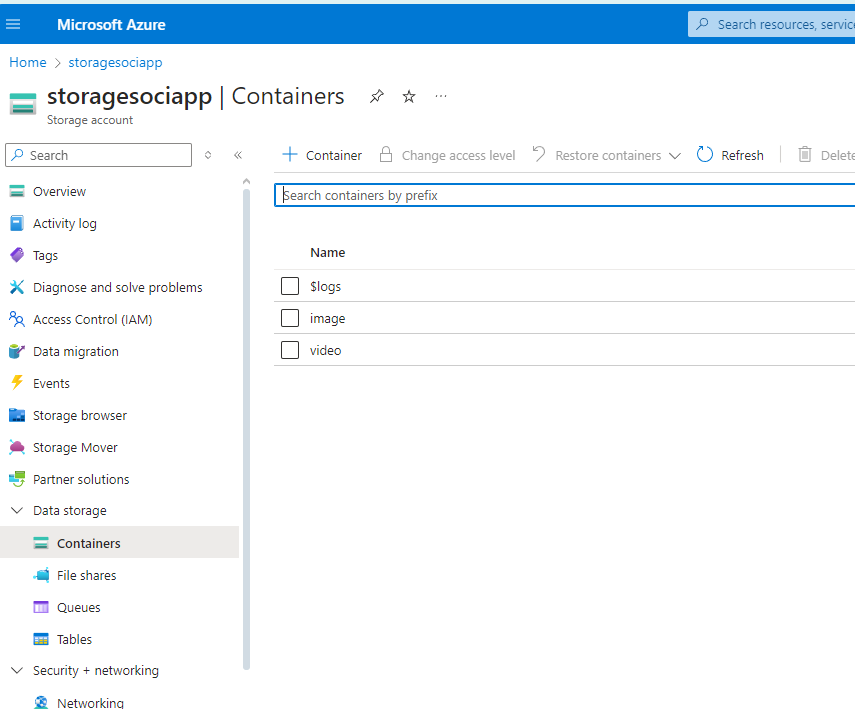


Dans Azure App Service, dans Setting > Environment variables.

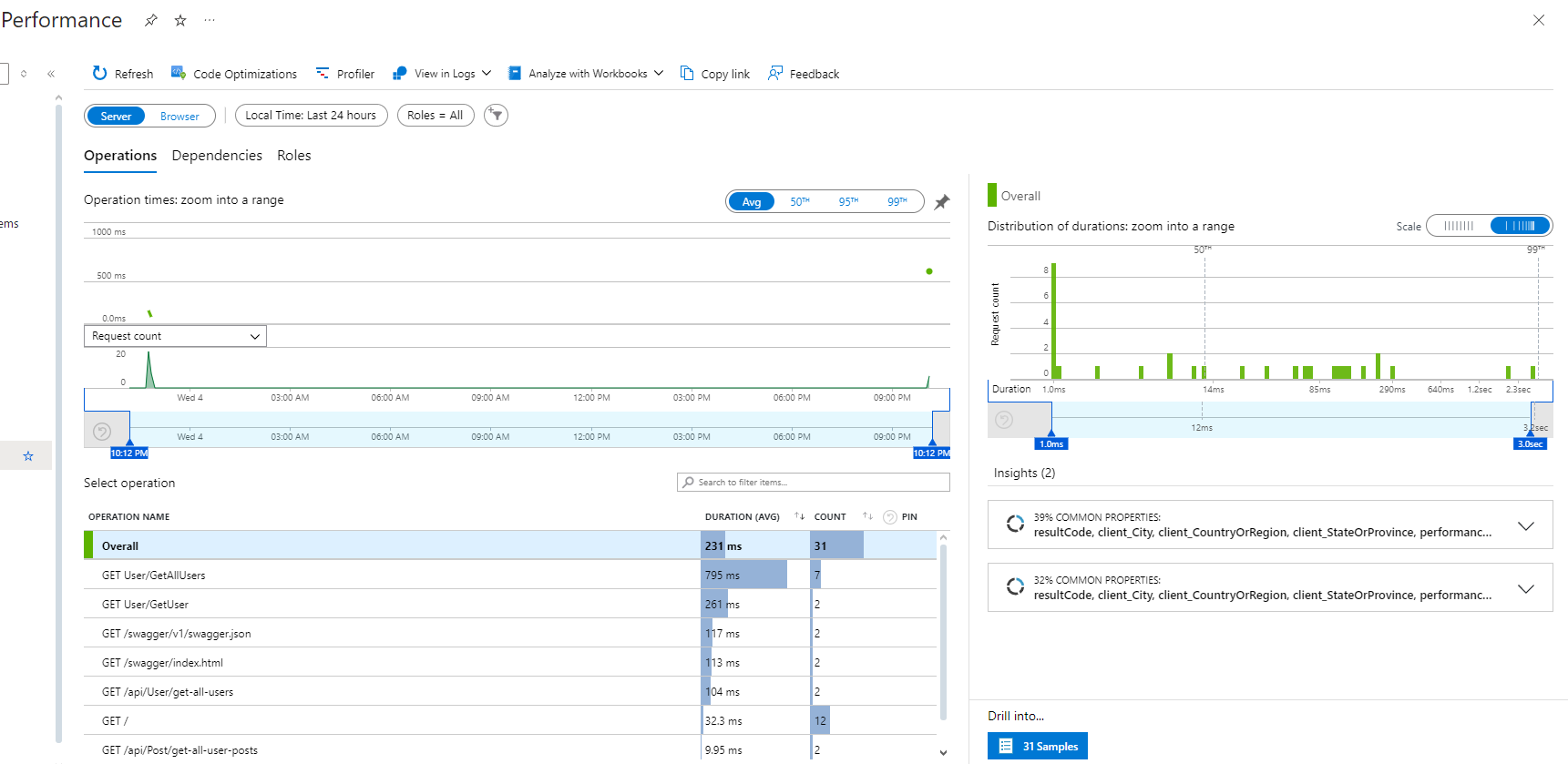
J’ai ajoute une variable d'environnement ConnectionString\_\_ConnectionDB avec la chaîne de connexion.

Storage :

Pour stocker les images et vidéos dans mon projet, j'ai choisi **Azure Blob Storage**. J'ai d'abord créé un **compte de stockage** et un **conteneur** sur Azure, puis ajouté la **chaîne de connexion** dans appsettings.json de mon projet. En utilisant le package NuGet  **.Storage.Blobs**, j'ai créé un service pour gérer l'upload des fichiers vers Azure. Ce service stocke les fichiers dans le conteneur et récupère l'URL du fichier, que j'enregistre ensuite dans ma base de données.

Dans l'API, j'ai ajouté une méthode pour recevoir les fichiers via **POST** et les stocker dans Azure. J'ai aussi configuré des **permissions de sécurité** pour contrôler l'accès aux fichiers. Après avoir testé l'upload, j'ai déployé l'application sur **Azure App Service**, offrant ainsi une solution fiable et sécurisée pour le stockage des fichiers multimédia dans l'application.

* Perfermance et scalability



Dans cette analyse de performances, je constate que mon application, via **Azure Monitor**, affiche les statistiques des requêtes reçues et traitées.

**Activité récente** : J’ai remarqué un pic de requêtes à **22h12**, ce qui peut indiquer une augmentation soudaine du trafic.

**Problèmes de performance** : L’API GET /User/GetUser est plus lente, comparée à d’autres endpoints qui répondent rapidement, comme GET /api/Post/get-all-user-posts

**Répartition des durées** : La majorité des requêtes sont rapides, mais certaines dépassent **3 secondes**, ce qui pourrait affecter l’expérience utilisateur.

Grâce à ces insights, je peux cibler les endpoints lents pour les optimiser, analyser les logs, et gérer les pics de trafic plus efficacement.

* Points améliorations

Je pourrais améliorer le projet en ajoutant des tests automatisés dans le pipeline, en utilisant Azure Key Vault pour mieux sécuriser les informations sensibles, et en optimisant la gestion des logs avec Application Insights

* Conclusion

Ce projet social, réalisé avec une API en .NET et un front-end en React, a été déployé avec les solutions Azure, telles que **App Service** pour l'API, **Azure Storage** pour les données non structurées et **Azure Database for MySQL** pour la gestion des données. J'ai automatisé le processus de déploiement via **GitHub Actions**.